Gramaglia, M., et al. Experimenting with SRv6: a tunneling protocol supporting network slicing in 5G and beyond. In, *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 14-16 September 2020 (Virtual Conference). IEEE, 2020, 6 Pp.*

DOI: https://doi.org/10.1109/CAMAD50429.2020.9209260

Experimenting with SRv6: a Tunneling Protocol supporting Network Slicing in 5G and beyond

Marco Gramaglia*, Vincenzo Sciancalepore†, Francisco J. Fernandez-Maestro‡,
Ramon Perez*§, Pablo Serrano*, Albert Banchs*¶
*Universidad Carlos III de Madrid, Spain †NEC Laboratories Europe, Germany ‡Ericsson Spain
§Telcaria Ideas, Spain ¶IMDEA Networks Institute, Spain

***Abstract*—With network slicing, operators can acquire and manage virtual instances of a mobile network, tailored to a given service, in this way maximizing flexibility while increasing the overall resource utilization. However, the currently used tunnelling protocol, i.e., GTP, might not be the most appropriate choice for the envisioned scenarios, given its unawareness of the underlay network. In this paper, we analyse the use of an alternative tunnelling protocol to transport user data, namely, Segment Routing IPv6 (SRv6). More specifically, we discuss its qualitative advantages, present a prototype implementation, and carry out an experimental comparison vs. GTP, confirming that it constitutes a valid alternative as tunnelling protocol.**

I. Introduction

The success of the upcoming 5th generation of mobile network (5G) and beyond is heavily tied with the implementation of a novel paradigm: *network slicing*. Strongly supported by the virtualization and programmability concepts, network slicing represents a turning point that allows network operators (NOPs) to acquire and self-manage a virtual instance of a mobile network to deliver diverse network services while increasing the overall resource utilization.

However, higher flexibility and augmented revenues might come at a price. Technical challenges have to be solved while installing network slicing along all network domains, such as (Radio) Access Network (R)AN, Transport and Core. Such a domains need a simultaneous and efficient interaction to properly provide service-level agreement (SLA) guarantees: this is accomplished by a novel architectural block, dubbed as Management and Orchestration (MANO), able to control, monitor and trigger actions onto each network (virtual) function. As a result, networks supporting slicing require advanced orchestration solutions that have attracted interest from both industry and academia showing advantages and drawbacks in current deployments.

Interestingly, the main standardization bodies (ISGs) have also listed network slicing in their priority activities. Recently, 3GPP has released the first official guidelines for 5G networks (Rel-15) wherein the network slicing concept has been unveiled associated with novel use cases and new standardization blocks. In particular, four different main classes have been defined: extreme/enhanced Mobile Broadband (x/eMBB), Ultra Low Latency and Reliable Communications (URLLC), massive Machine Type Communications (mMTC) and (recently) Vehicular-to-Infrastructure communication (V2X) [1]. Additionally, specific core network functions and procedures have been introduced to correctly manage network slice lifecycle management operations, such as Network Slice Selection Function (NSSF), Network Slice Selection Assistance Information (NSSAI), and so on.

While this effort usually focuses on the control plane and management functions, most of the network-slicing-related researches from the user plane perspective are targeting issues at the RAN level. Indeed, achieving flexible network slicing in the network access is fundamental to correctly route "sliced" network flows and to transport service-related (and isolated) information to the intended core instance in an efficient manner.

In this context, the GPRS Tunnelling Protocol (GTP) currently used in mobile networks might not result the best fit to support sliced networks. The reasons are as follows. GTP consists of control-plane (C-plane) and user-plane (U-plane) sections, where the former signals movements of a user and establishes user-plane data paths by means of tunnels between the end user and an anchor-node (over IP-based backhaul network). This reduces the complexity as the control-plane does not require communication with the underlay networks (such as MPLS label-switched paths, VLANs, L3-VPNs and so on). However, when network slices are in place, the orchestration solution should be aware of the underlay networks, to guarantee the expected SLAs from the U-plane perspective.

To solve the above problem while keeping the network deployment simple, it would be recommendable to integrate the U-plane into a layer that forwards packets through requested data-paths (note that GTP do not support such operation). The most appropriate network layer able to integrate all the required functions is the end-to-end IP layer. More specifically, an IPv6-enabled layer might play a relevant role in this context, given the "limitless" amount of available address space that can be used for multiplexing mobile sessions between tunnel endpoints of the user-plane.

The IPv6 layer might also integrate network functions into it—as specified by the IETF work on *Segment Routing IPv6* (SRv6) [2]—thereby enabling that forwarding functions are represented as IPv6 addresses. SRv6 network programming might also define different forwarding functions, such as, e.g., encapsulation or decapsulation and routing while comprising VPNs, traffic engineering, and service chaining aspects. Ad-

ditionally, SRv6 might also compose data paths in the end-to-end IPv6 address layer by means of network programmability. This adds flexibility as mobile applications only need a single IPv6 layer to run on, for e.g. network resources and optimized paths for low-latency (and high-reliability) may be represented as an IPv6 address. Once they have been abstracted as IPv6 IDs, the C-plane could comprise of slices with them as nodes of the U-plane function (UPF), and deploy required data-paths for mobile applications on them [3].

In this paper, we make the case for the use of SRv6 as a transport protocol in network slicing environments, which include 5G and beyond networks. More specifically, we make the following contributions:

- In Section II, we summarize and compare GTP and SRv6, discussing the qualitative advantages of the latter over the former.
- In Section III, after an overview of the existing alternatives to implement SRv6, we present our implementation based on the P4 programming language.
- In Section IV, we present a quantitative performance evaluation of SRv6, comparing its performance vs. GTP in a number of scenarios.

Finally, we summarize the main results and sketch new lines for future research in Section V.

II. Tunneling Protocols for User Data traffic in 5G Networks

The tunnelling protocol design is of paramount importance to properly run network slices. Hereafter, we overview the overall Rel-15 architecture to illustrate the main interfaces that have been disclosed by the recently published standard guidelines. We then summarize the existing GTP properties while shedding the light on the main novelties introduced by the Segment Routing IPv6 (SRv6) protocol.

A. 3GPP UPF reference points

We summarize in Fig. 1 the 3GPP-related reference points of the User Plane Function (UPF), which are illustrated along with other elements of the 5G architecture [4]. The User Equipment (UE) establishes a data session and gets assigned the corresponding User Plane Function (UPF) in the core network, responsible of routing and forwarding the traffic back and forth between the UE and a Data Network (DN). The UPF selects the appropriate transport network for the user traffic by means of N4 reference point, which provides an interface with the Session Management Function (SMF). This interface enables selecting a Network Instance ID based on the S-NSSAI of the PDU session.[1] Considering the actual transmission of user data traffic, this arrives from the gNB to the UPF via the N3 interface, while the UPF is connected to the DN via the N6 interface. Finally, the interface N9 is defined in case there is communication between UPFs (e.g. from an intermediate to an anchor UPF).

[1]Note that there is not a one-to-one mapping between network slices and transport network slices, and that there are other mechanisms to select the Network Instance ID

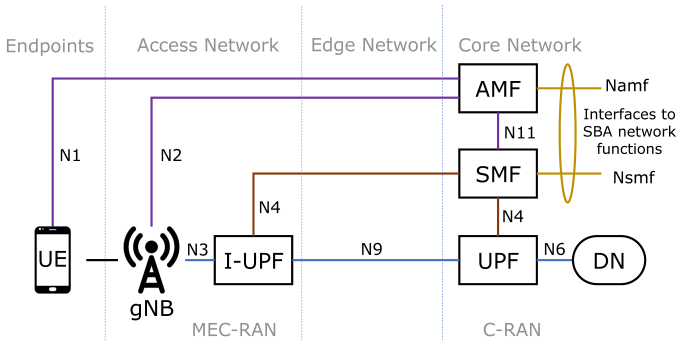

Fig. 1. Key components of the data plane in a 5G network.

B. GTP-U encapsulation

The existing encapsulation process is called GTP encapsulation and is the one currently used in mobile networks (e.g., Gn and Gp interfaces in GPRS, Iu, Gn, and Gp interfaces in UMTS). It has been proposed for 5G Systems as well, i.e., for the interfaces described above and the Xn interface (i.e., the one between RAN nodes, which is not considered in this paper). According to its specification [5], the encapsulation consists of adding a GTP-U header to the IP datagram (*i.e.*, the T-PDU) and using UDP/IP as transport layer. Both IPv4 and IPv6 traffic types are supported.

The GTP-U header, as illustrated in Fig. 2, has a variable length, depending on the number of optional fields. The minimum length is 8 bytes and includes the Tunnel Endpoint Identifier (TEID), which is the key variable towards forwarding as this element is used by the receiving end of the packet to identify the PDP context of the T-PDU. As already discussed in Section I, this protocol does not provide native path differentiation, and it is already falling short for some of the use cases envisioned by 5G (we note that the first version of GTP-U was standardized in [5], more than a decade ago).

C. Segment Routing v6

Segment Routing for IPv6 uses an IPv6 Routing Header Extension, the Segment Routing Header (SRH), defined in [6]. Its structure is depicted in Fig. 2 and consists on the `Segment Lists` (SL), in which each segment is identified with an IPv6 address, *i.e.*, the SegmentID (SID). The current active segment is the destination address of the packet, while the next address is indicated by the Segments Left field of SRH.

The use of Segment Routing enables network programmability, as one can associate a given SID with given processing function behaviors (in [7] and [8], several basic examples are defined, and in [3], it is also defined specific examples for the user data plane of mobile networks). These functions can be classified into two main categories: *endpoint behaviors*, which have a Local SID to the destination address received, being associated to a processing function in the local node, and *transit behaviors*, in which there is no local SID associated to the destination address of the packets received, so they are not bound to a SID, corresponding then to either SRv6 source

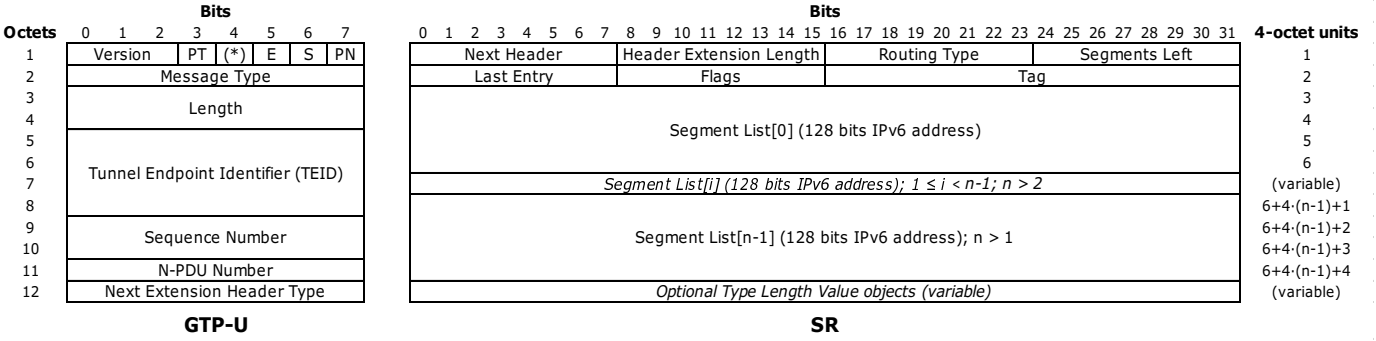

Fig. 2. GTP-U (left) and SR (right) headers.

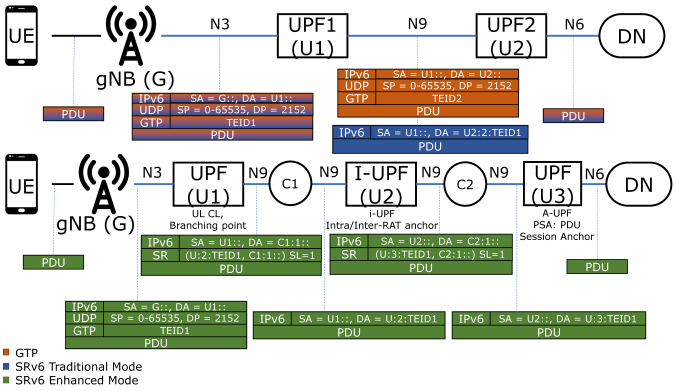

Fig. 3. Uplink packet flow. Top figure: using GTP (orange) or SRv6 in Traditional Mode (blue). Bottom figure: SRv6 in Enhanced Mode.

nodes that are introducing the SRH in the packets or SRv6 transit nodes steering traffic.

Depending on whether it is considered as a simple replacement for GTP-U, or instead a step towards the adoption of network programmability, one can distinguish between two modes of SRv6 operation in a mobile network [9] (note that, in both cases, the network should be SR-aware):

- **Traditional mode:** With this mode, there is a direct mapping between GTP-U and SRv6 encapsulations. It does not introduce changes to the network architecture and the impact is therefore kept minimum. This mode would be used, for instance, in the topology illustrated in Fig. 3 (top), where the user packet transverses four hops to arrive to the DN.
- **Enhanced mode:** By introducing more than one SID in the SRH (*i.e.*, a packet can traverse more nodes than in the Traditional mode), the use of more advanced functions in the network path of the traffic user flows is allowed, enabling to steer traffic in this way by using different paths and supporting more sophisticated behaviors, e.g., service programming. Thanks to this SRv6 mode, one can associate non-3GPP services, such as Traffic Engineering in nodes along the path, as illustrated in Fig. 3 (bottom).

D. Scalability and flexibility in SRv6-based networks

Segment routing supports performing source routing for traffic engineering without incurring in the heavy workload of states and signalling in the nodes by using IDs, such as MPLS labels or IPv6 addresses. In contrast, GTP is mainly coupling the mobile infrastructure to the underlying transport network representing a threat to the status-quo, as it leans towards a single-vendor monolithic mobile network implementation.

Adopting SRv6 in the mobile infrastructure brings an added-value as the operator may employ the same Layer-3 VPN techniques that will be used for managed and deliver enterprise services, which represent a valuable business change in fixed mobile convergence (FMA) and 5G fixed mobile access (FMA). In addition, SRv6 will help to boil down the unnecessary complexity by reducing protocol overheads while completely eliminating the need of the UPF to keep states. This would noticeably improve the scalability (for e.g. in cloud native applications). As outlined within the IETF draft document, SRv6 allows efficient SLA-enabled multi-cast content injection by means of the standard unicast core with a novel method, namely “Spray.”

However, SRv6 will also show its drawbacks when dealing with legacy suppliers as a packet processing engine should continually dig deep into the IPv6 header. Essentially, SRv6 may require a programmable software data plane to be effectively implemented, UPF as fully virtualized network function, and specialized data plane acceleration solutions to continuously meet deep packet inspection requirements.

Given the amount of advantages as well as the main technology limitations, the 3GPP has started a study item “Study on User Plane Protocol in 5GC” to find candidates for updating the user-plane protocols within Rel-16 of 5G phase-2: *SRv6 is a reasonable option to evaluate*.

III. Implementation

Here we describe our experimental methodology based on open source software. All the code used in this experimentation is available online.[2]

[2]https://github.com/wnlUc3m/slicing-srv6

A. Existing implementations of GTP and SRv6

To evaluate the performance of a SRv6 implementation of the 5G Core Network internal interfaces, we had to analyze several components, as discussed in the following.

a) GTP-U: Besides the proprietary implementations of the major network component manufacturers, the main open source solution for the GTP-U implementation is the one available in the Linux Kernel, from v4.7.0. This implementation [10], contributed to the Linux Kernel by osmocom, can be compiled as an external module. However, as the 3GPP standard [5] sets IPv4 as the required protocol to be supported by any implementation of the GTP-U protocol, IPv6 is left as a recommendation. As a result, the current Linux Kernel implementation of the GTP-U does not support IPv6 neither as inner nor as outer protocol [11].

b) SRv6: There is a wider landscape of open source alternatives for implementing SRv6. The recent versions of the Linux Kernels (starting from v4.10) already support the latest definition of the SRv6 protocol as defined in [7] and [8] and, additionally, the `srext` module [12] is also available for experimentation. Besides pure Kernel implementation, also the Fast Data Project (FD.io) [13] implements several of the SRv6 behaviors. However, the lack of integration with the kernel stack indeed provides very high performance for data packets processing, but it also lock-in the packet forwarding technology to the hardware platform.

To address this possible issue and allow a more modular implementation of the network functionality, starting from Linux v4.18, there is also an available SRv6 implementation [14] based in extended Berkeley Packet Filter (eBPF) [15]. By leveraging the advanced features of eBPF (which can inject arbitrary code down into the kernel stack to achieve a sort of per packet processing), this implementation can offer specific SRv6 behaviors for every handled packet. In the specific case of SRv6, the eBPF implementation handles packets that have SRH with segments left (SL) greater than 0.

B. Implemented solution

Given the limitation of the solutions described above, we decided to follow a completely software defined approach for our implementation. More specifically, the lack of IPv6 support for GTP-U in the kernel forced us to implement both SRv6 and GTP-U on top of behavioral model software switch (or `bmv2`) [16] to allow a fair comparison between the two solutions. The encapsulation and decapsulation methods where implemented thus in P4 [17], using the SRv6 interface [18]. The software has been modified to adding new tables and actions as needed.

Then `bmv2` switches could be assigned to a specific network interface of the host, with a specific P4 configuration produced by the P4 compiler. To achieve flexibility in the configuration, we injected the specific runtime configuration trough a Command Line Interface (CLI) connects to a Thrift RPC server[3] running in each switch process.

[3]https://thrift.apache.org

The original `bmv2` prototype contained three tables: `fwd`, `gtp_v6`, and `srv6_localsid`, that are executed in a pipeline. Each table has some matches associated and, hence, actions associated to the match. Thus, the code has been extended to include new actions for reduced SRv6 encapsulation, where only one destination SID is included, so the P4 parser was be extended to allow IPv6 over IPv6 encapsulation without having an SRH (when using reduced encapsulation and only one segment in the SRL).

Also, we implemented new actions to change from one GTP encapsulation with certain TEID to another tunnel TEID encapsulation. Finally, we extended the original P4 code to handle the switch from one GTP encapsulation to another GTP one, or the enforcement of an SRv6 action.

IV. Performance Evaluation

A. Testbed setup

Given that our performance evaluation considers a number of different network topologies, the testbed setup has been done by deploying Vagrant boxes[4] (using underlying VirtualBox[5] as virtualization provider). This facilitates the setup and connectivity of several VMs for the different testbeds using the different configuration files options and provisioning scripts. The host running the VMs is equipped with i7-8650 CPU at 1.90GHz and 32 GB of RAM. Each VM is executed with 1 virtual CPU core and 2 GB of RAM.

B. Throughput performance in multi-slice scenario

To evaluate and compare the potential differences in throughput performance between GTP and SRv6, we focus on the uplink in a multiple slice scenario that contemplates the different deployment options studied, and which is presented in Fig. 4. The scenario consists of an UPF (UPF #1), shared among two slices, receiving GTP traffic from a gNB classified in two different tunnel IDs, where each tunnel (#1 and #2) identifies the traffic that belongs to a specific slice (A and B, respectively). Depending on how the traffic is handled from the UPF #1 in the uplink flow, according to the protocol used, it has been defined three different deployment options, also reflected in Fig. 4 with three different colours:

a) GTP-U (orange): this option is based on the traditional GTP protocol, where the traffic received in UPF #1 (from tunnels #1 and #2) is routed into different GTP tunnels (#3 and #4, respectively) through the N9 interface towards specific UPFs (#2 and #3) assigned to each slice (A and B). These UPFs then de-encapsulate the user plane traffic in order to send it to the corresponding DNs (#1 and #2). No encapsulation is performed in the N6 interfaces.

b) SRv6 Traditional Mode (blue): this case contemplates a gradually migration from GTP to SRv6 in user plane by implementing the SRv6 Traditional Mode as a direct alternative to GTP, with a mapping between TEID and SRv6 once the GTP traffic arrives to UPF #1. It has the same logical

[4]https://www.vagrantup.com

[5]https://www.virtualbox.org

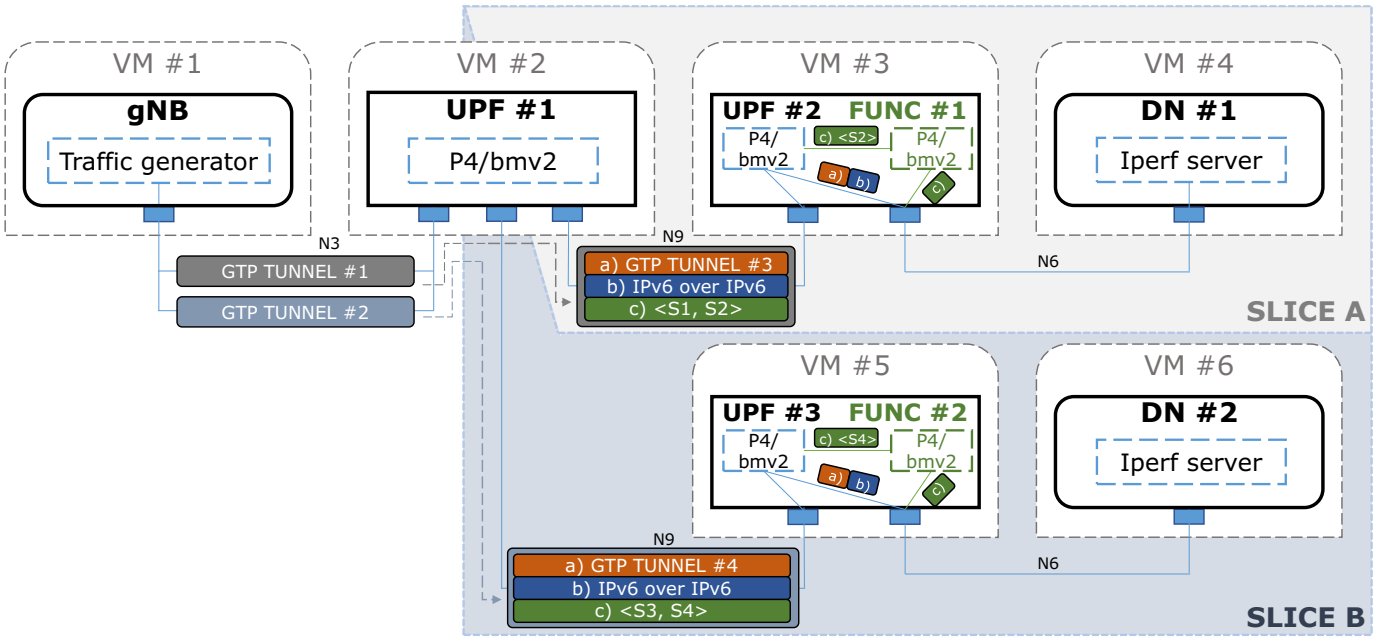

Fig. 4. Multiple slice scenario with different choices for the N9 user plane: (*a*) GTP, (*b*) SRv6 Traditional Mode, and (*c*) SRv6 Enhanced Mode.

blocks as the previous option, but also includes in UPF #1 the End.M.GTP6.D behavior, which is the "Endpoint function with IPv6/GTP decapsulation into SR policy," defined in [3] (*i.e.*, it de-encapsulates incoming traffic coming in GTP and it encapsulates with SRv6)). More specifically, only one segment is used in the SRv6 traffic, so after reaching the next hop in the uplink flow (UPF #2 and #3), the SRH and outer IP headers are removed in the traffic that is delivered to the corresponding DNs (#1 and #2), according to [9].

c) SRv6 Enhanced Mode (green): this last option gives the more flexibility and network programmability capabilities, where the traffic is steered into different slices and, inside each slice, with different SRv6 policies. Moreover, it includes some additional nodes that can be either non-3GPP entities or functions, and also implements different behaviors for each of the SIDs in the segment list of the SRH. In this case, the SRv6 traffic in N9 interface contemplates two segments (<S1, S2> in slice A, <S3, S4> in slice B), and the next-hop VMs (#3 and #5, respectively) have been extended by including a new P4 `bmv2` (displayed with a different colour in Fig. 4, according to the corresponding slice), identified as FUNC nodes (#1 and #2), where packets are delivered. As a result, the first segments (S1 for slice A, and S3 for slice B) correspond to the next UPFs (#2 and #3, respectively), and the second ones (S2 for slice A, and S4 for slice B) correspond to FUNC #1 and #2, respectively (these segments are also displayed under each UPF-FUNC connection in Fig. 4). As a result, the segment lefts field in SRH is being decreased in each hop, and finally the FUNC entities are the latest ones in the segment list.

In this kind of multi-slice scenario, the use of SRv6 as an alternative to GTP protocol in user plane presents a number of advantages, due to its flexibility and the possibility to program different behaviors for the network and slices in an easier way (as compared vs. GTP). In what follows, we test these advantages against the throughput obtained for the three configurations. To this aim, for simulating UE traffic data coming from gNB, it has been generated GTP traffic with a Python script elaborated with `scapy` libraries [6], which is then injected in N3 interface at different rates with the `tcpreplay` utility [7] through a GTP tunnel as transport protocol.

In Fig. 5, we plot the Packet Delivery Ratio (PDR) as a function of the injected throughput, which is generated using the `iperf` tool.[8] As the figure illustrates, for low generation rates the PDR is 1 for all cases, but as the rates increases, the network has some issues to transport the traffic, and losses appear. More specifically, the figure illustrates that the appearance of the losses depend on the configuration configured: the SRv6 Traditional Mode provides the best performance, with the highest values in terms of PDR, particularly at higher rates, which is when the computational differences are more noticeable.

Next in performance comes the traditional GTP case, which has a similar performance to the previous one, but has a notable PDR drop around 30 Mbps. Finally, the worst performing scheme is the SRv6 Enhanced Mode, which starts suffering from losses at approx. 20 Mbps –still, note that the PDR is 1 for all values below that threshold. In this way, we have identified the cost of the more flexible operation introduced by the SRv6 Enhanced Mode: this configuration enables new paths for the traffic, being able to steer and

[6]https://scapy.net
[7]https://tcpreplay.appneta.com
[8]https://iperf.fr

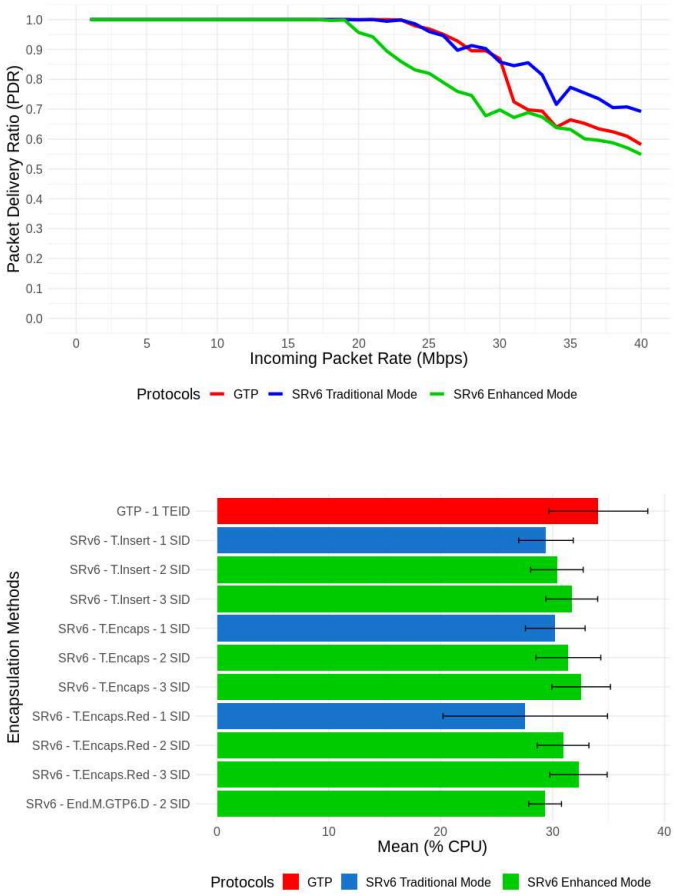

Fig. 6. Total percentage of CPU usage for different encapsulation methods.

perform additional functions that are not possible with GTP or SRv6 Traditional Mode (which is basically a 1-to-1 mapping of the GTP scenario). As a result, this extra functionalities and flexibility for adding new functions dynamically in the slices gives an added value which is not present in the legacy networks based on GTP, but causes a small performance reduction in the maximum traffic that can be supported (in our scenario).

C. Resource consumption

One of the main advantages of the SRv6 approach is the computational efficiency. As the path information is indicated upfront, less effort is required to achieve this information and perform switching compared to the GTP-U approach. Therefore, we evaluate this aspects with a simple source - sink scenario with two VMs running the `bmv2` P4 implementation discussed before. Then, we measured the CPU consumption using the Linux `top` command when generating 10 Mbps UDP/IPv6 traffic of 100 kbytes jumbo datagrams. As shown in Fig. 6, on average the SRv6 implementation in all of his flavours always uses less resources than the GTP-U one, showing the validity of this approach also from the infrastructure optimization point of view. The figure also suggests that the number of segments has an impact on performance, an analysis that we will perform in the future work.

V. Summary and Future work

In this paper, we have discussed the advantages of Segment Routing v6 (SRv6) as a tunnelling protocol for upcoming 5G Core Networks. In particular, we envision SRv6 as the replacement for the well-accepted GTP-U tunneling protocol in the user plane. GTP-U is used to carry user data in existing mobile networks by creating tunnels per session. However, such tunnels are established between anchor nodes that may connect different network domains within a rigid framework. This prevents the network operator from efficiently optimizing the data paths. As proved, SRv6 can make the transport much simpler. With a SID field, tunnel endpoints identifiers (TEID) can be easily encoded within the protocol stack. In addition, SRv6 can also replace underlay transport layers, such as MPLS or L2-tunneling protocol allowing for the introduction of SRV6/IPV6 as the only valid transport layer in 5G and beyond networks.

Acknowledgements

This work has been supported by the European Commission and the 5G-PPP H2020 Programme under the grants 825012 (5G-CARMEN), 815074 (5G-EVE) and 856950 (5G-TOURS).

References

[1] 3GPP, "Summary of rel-15 work items (release 15)," TR 21.915, v15.0.0, Oct. 2019.
[2] E. C. Filsfils, "Segment routing architecture," RFC 8402, Jul. 2018. [Online]. Available: https://datatracker.ietf.org/doc/rfc8402/
[3] S. Matsushima, "Segment routing IPv6 for mobile user plane," draft-ietf-dmm-srv6-mobile-uplane-06, Sep. 2019. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-dmm-srv6-mobile-uplane/
[4] 3GPP, "System architecture for the 5G system (5GS)," TS 23.501, v16.2.0, Sep. 2019.
[5] ——, "General packet radio system (GPRS) tunneling protocol user plane (GTPv1-U)," TS 29.281, v15.6.0, Sep. 2019.
[6] E. C. Filsfils, "IPv6 segment routing header (SRH)," draft-ietf-6man-segment-routing-header-24, Oct. 2019.
[7] ——, "SRv6 network programming," draft-ietf-spring-srv6-network-programming-03, Sep. 2019.
[8] ——, "SRv6 NET-PGM extension: Insertion," draft-filsfils-spring-srv6-net-pgm-insertion-00, Sep. 2019. [Online]. Available: https://datatracker.ietf.org/doc/draft-filsfils-spring-srv6-net-pgm-insertion/
[9] 3GPP, "Study on user-plane protocol in 5GC," TR 29.892, v16.0.0, Sept. 2019.
[10] Osmocom, "Linux kernel GTP-U implementation." [Online]. Available: https://osmocom.org/projects/linux-kernel-gtp-u/wiki
[11] "The linux kernel GTP tunneling module." [Online]. Available: https://www.kernel.org/doc/Documentation/networking/gtp.txt
[12] "srext - a linux kernel module implementing SRv6 network programming model." [Online]. Available: https://netgroup.github.io/SRv6-net-prog/
[13] "Fd.io. the world's secure networking data plane." [Online]. Available: https://fd.io/
[14] "Programming network actions with BPF." [Online]. Available: https://segment-routing.org/index.php/Implementation/BPF
[15] "BPF and XDP reference guide." [Online]. Available: https://cilium.readthedocs.io/en/latest/bpf/
[16] "Behavioral model repository." [Online]. Available: https://github.com/p4lang/behavioral-model
[17] P. Bosshart *et al.*, "P4: programming protocol-independent packet processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, pp. 87–95, July. 2014.
[18] "p4srv6. proto-typing SRv6 functions with P4 lang." [Online]. Available: https://github.com/ebiken/p4srv6